Theater Management System

This project is submitted as part of course requirements for CS251, Database Systems.

Submitted by

Feyaz Baker 181CO119

Nihar KG Rai 181CO235

The source code is available at: <https://github.com/NegaMage/TheatreManagement>

**Table of Contents**

[Introduction 3](#__RefHeading___Toc401_1965992706)

[System analysis 4](#__RefHeading___Toc403_1965992706)

[Existing systems 4](#__RefHeading___Toc405_1965992706)

[Proposed System 4](#__RefHeading___Toc407_1965992706)

[Feasibility study 6](#__RefHeading___Toc409_1965992706)

[1. Technical feasibility 6](#__RefHeading___Toc411_1965992706)

[2. Economic feasibility 6](#__RefHeading___Toc413_1965992706)

[ER Diagram 7](#__RefHeading___Toc415_1965992706)

[Images 8](#__RefHeading___Toc417_1965992706)

[Implementation 15](#__RefHeading___Toc419_1965992706)

[Why Django 15](#__RefHeading___Toc421_1965992706)

[How it was done 15](#__RefHeading___Toc423_1965992706)

[Normalization 15](#__RefHeading___Toc425_1965992706)

[A Deep-dive into the technical side 17](#__RefHeading___Toc427_1965992706)

[Model 17](#__RefHeading___Toc429_1965992706)

[View 17](#__RefHeading___Toc431_1965992706)

[Controller 17](#__RefHeading___Toc433_1965992706)

[Requirements 18](#__RefHeading___Toc435_1965992706)

[Conclusion 18](#__RefHeading___Toc437_1965992706)

# Introduction

Today’s world is dominated by media and theater giants. It is very rare to see small companies trying to enter the theater market. Such large companies normally have trouble streamlining the flow of data; specifically, the data pertaining different movies in different regions. Such streamlining will also assist customers in comparing the prices for different movies and make a consumer-friendly choice in one place.

Our software allows different theater corporations to advertise their movies as well as allowing customers to make a decision on which movie to watch without too much hassle. The software has 2 main modules:

* A customer module
* A management module

The software allows different corporations create their management portal to showcase their movies and update features such as spaces etc. The users are able to make their own personal account and browse through the available movies and buy tickets for the ones they are interested in.

# System analysis

## Existing systems

System Analysis is a detailed study of the various operations performed by a system and their relationships within and outside of the system. Here the key question is- what all problems exist in the present system? What must be done to solve the problem?

Analysis is key in order to determine if our proposed software is going to be useful; we begin by looking at the data collected through tools like data flow diagrams and interviews. Training, experience and common sense are required for collection of relevant information needed to develop the system. This can then be used to make well informed decisions on what our system needs to solve.

System analysis is conducted with accordance to this check list

1. Initial investigation
2. Information gathering using the above techniques
3. Applying the tools for a structured analysis
4. Feasibility and improvement analysis

Currently there are already websites for theaters that display the currently showing movies and give details for booking tickets. The difficulty most users have involve finding the prices for different showings without the hassle of checking multiple websites. Additionally, lesser known cinema giants and new movie releases struggle to emerge onto the market which generates a monopoly for such companies negatively affecting consumers.

## Proposed System

Our proposed system allows any Cinema company to create their own admin account which allows them to:

* Add showing movies
* Set ticket pricing
* Add the number of free seats
* Edit movie rating
* Fix their own show times
* Create private screenings/early screenings
* Ensure a free and fair market.

For Users the system is aimed at making the situation of booking movie tickets as hassle free as possible it involves:

* User account creation
* View the available movies
* Booking tickets

The system aims to be user friendly for both involved parties and requires minimal training on the part of companies to implement it.

This proposed system will act as a marketplace and also remove the issue of building their own website for cinema companies.

During a time like this with the pandemic this can be a very easy way for companies to direct users to make a regulated number of customers in to avoid contact between them.

# Feasibility study

A key concept that is required when designing any software product is its feasibility. Feasibility is the study of impact, which happens in the organization by the development of a system. The impact can be either positive or negative. When the positives outweigh the negatives, then the system is considered feasible. The idea of feasibility can be broadly classified into 2 categories:

### Technical feasibility

This involves the difficulties in bringing the project into the software space. For our task of generating a database system to manage theaters it can be assumed that the project is technically feasible. However, for official use we will need a host to utilize the service to its full extent.

The only other technical requirement would be the integration of a payment portal through third party payment gateway sites which has not been implemented in the current iteration of the software. Such integration can be made easily with processes like PayUMoney, PayPal, TechProcess or any online payment giant, but we couldn’t integrate without submitting legal documents and setting up an actual bank account.

### Economic feasibility

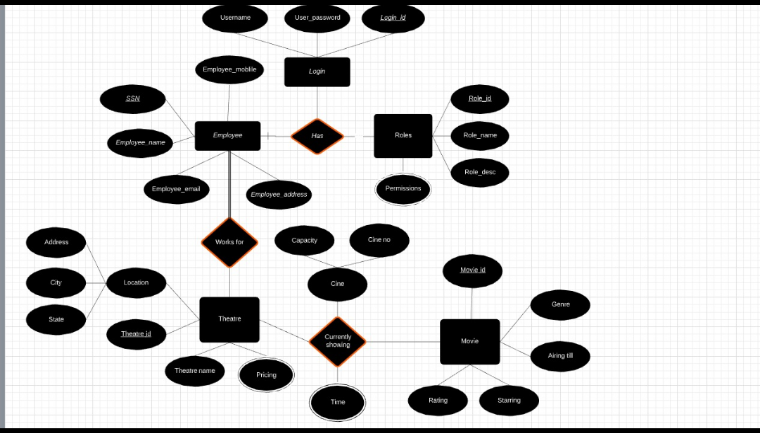
This is a more nuanced topic as when it comes to economic feasibility a lot of variables are involved and intense data gathering is required to fully comprehend the economic feasibility of a product. That being said we can generate a general idea on how the introduction of the product may affect the general market.

The basic concept would be that this product would ease entry for companies into the cinema market additionally it allows users to compare prices and pick the choice that satisfies their need for minimum cost thus increasing competition in the market. This in turn would lead to a fall in the demand for overpriced theaters bringing them in line with industry standards and overall being a major asset for any consumer.

The above statement may raise the fear that major companies would be unwilling to invest however due to the fact that our website acts as a free source of advertising they too would be compelled to utilize our service thus further empowering the stance of our service.

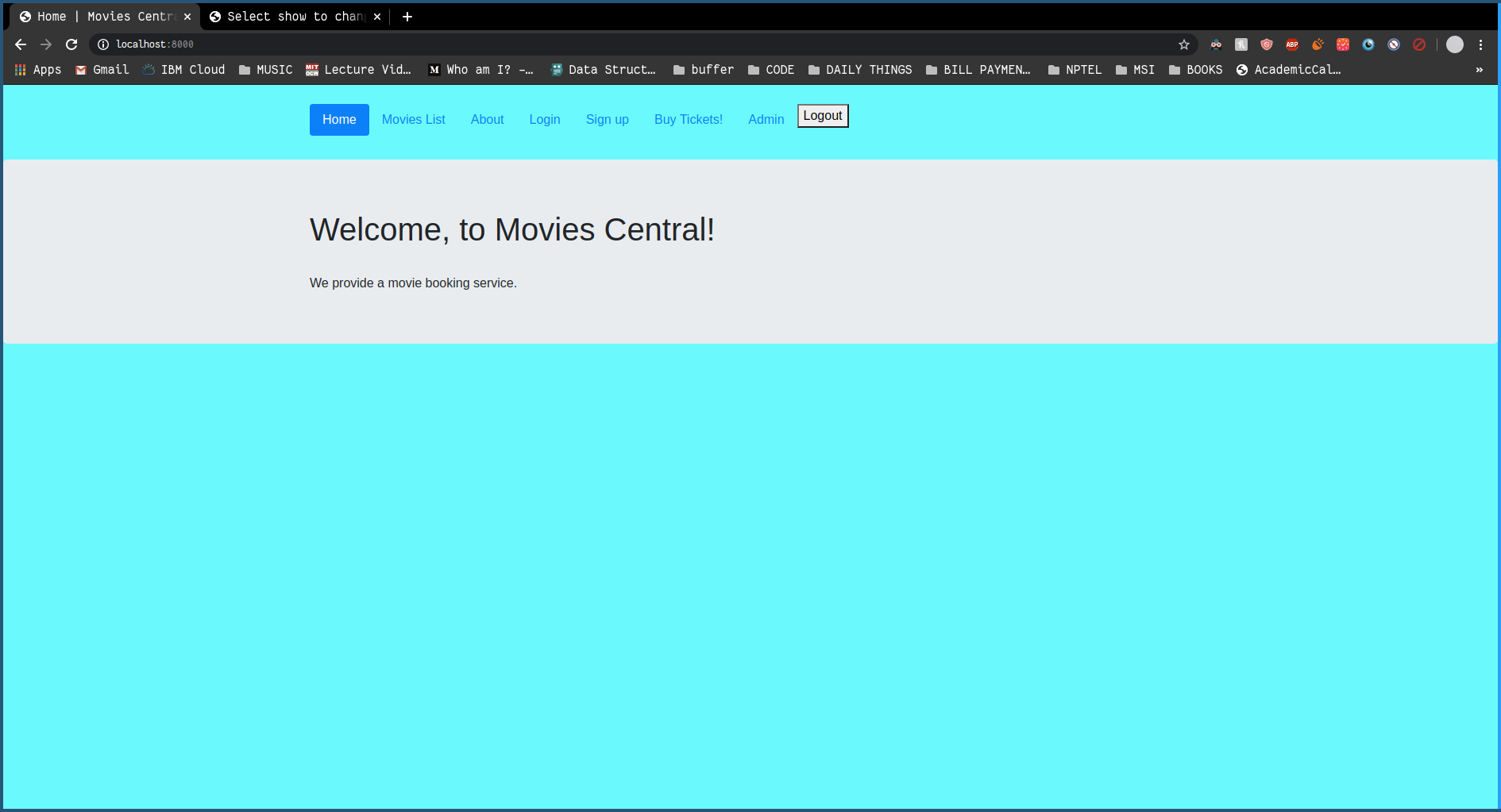
Due to these factors we can conclude that this service would be economically feasible.

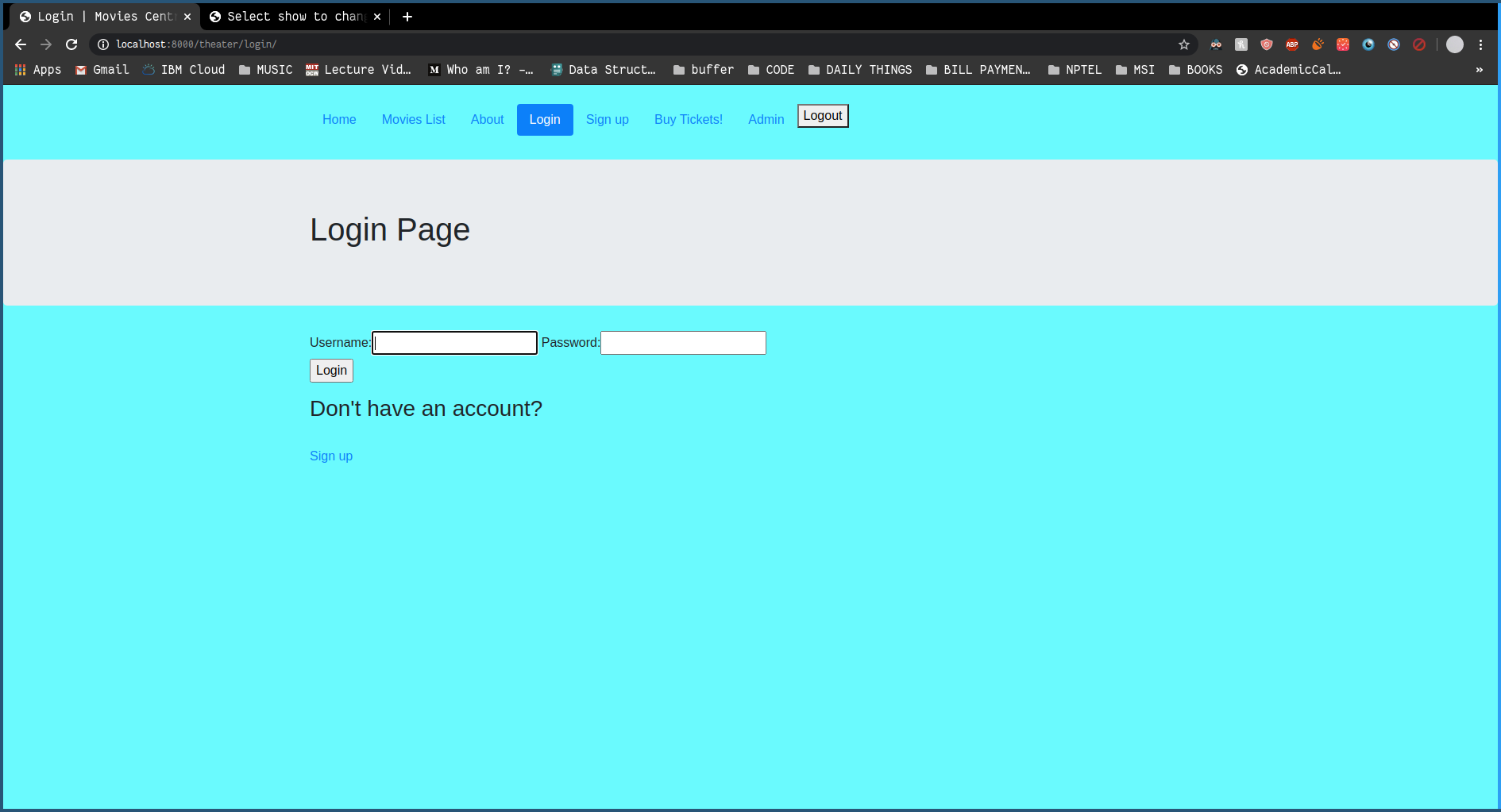
# ER Diagram

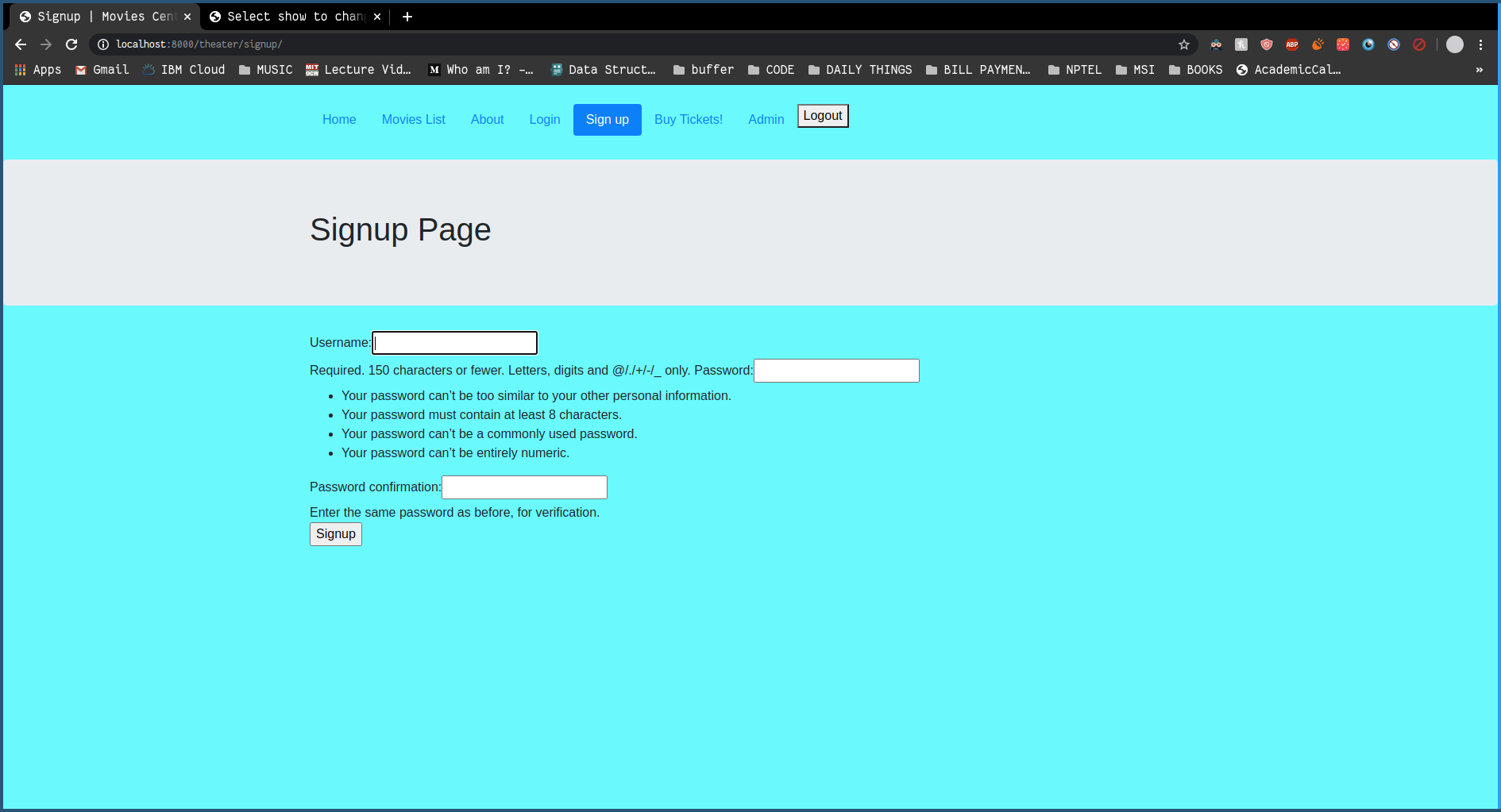


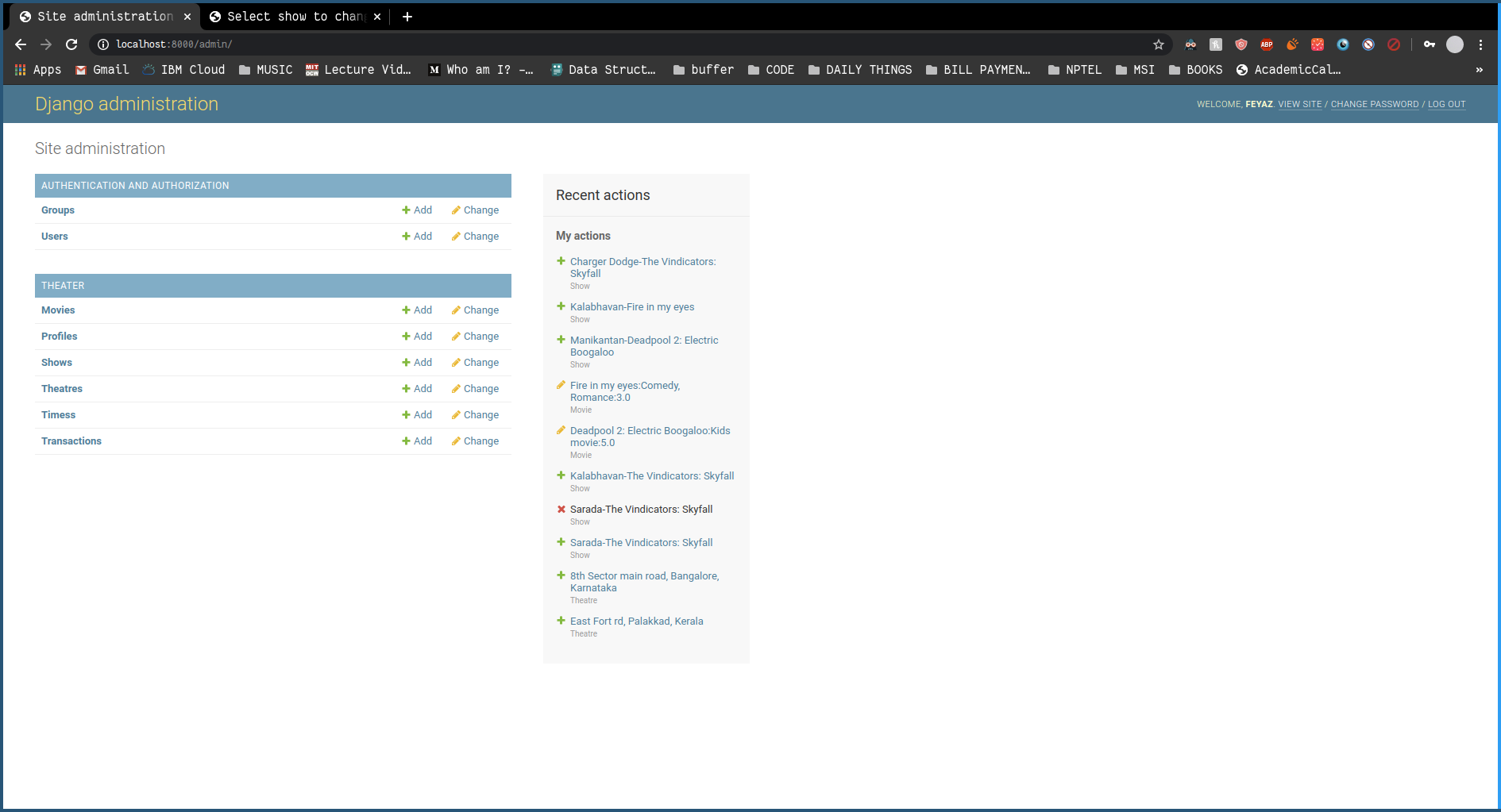
The above was the originally proposed ER model that we would follow for our database.

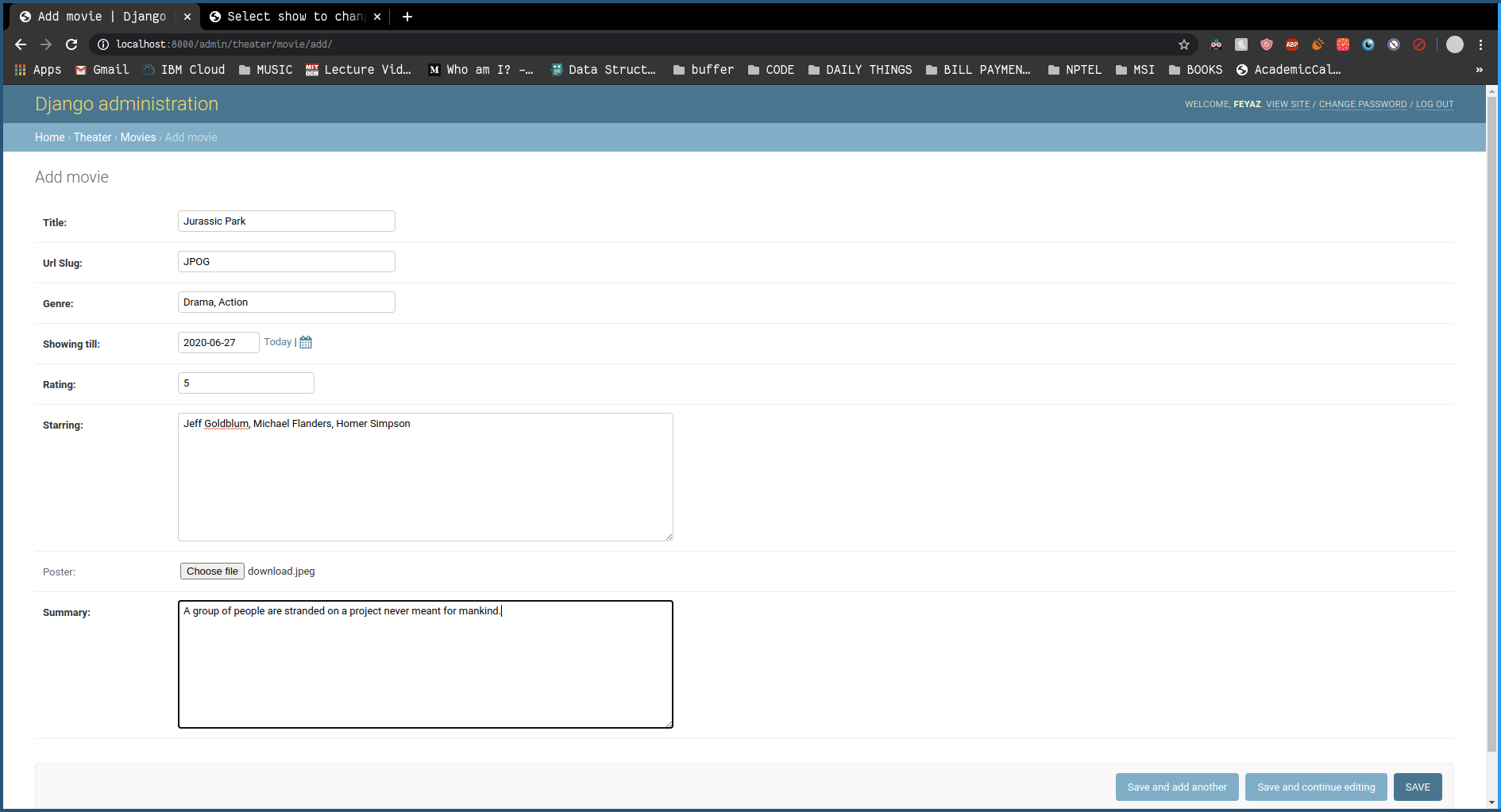
# Images

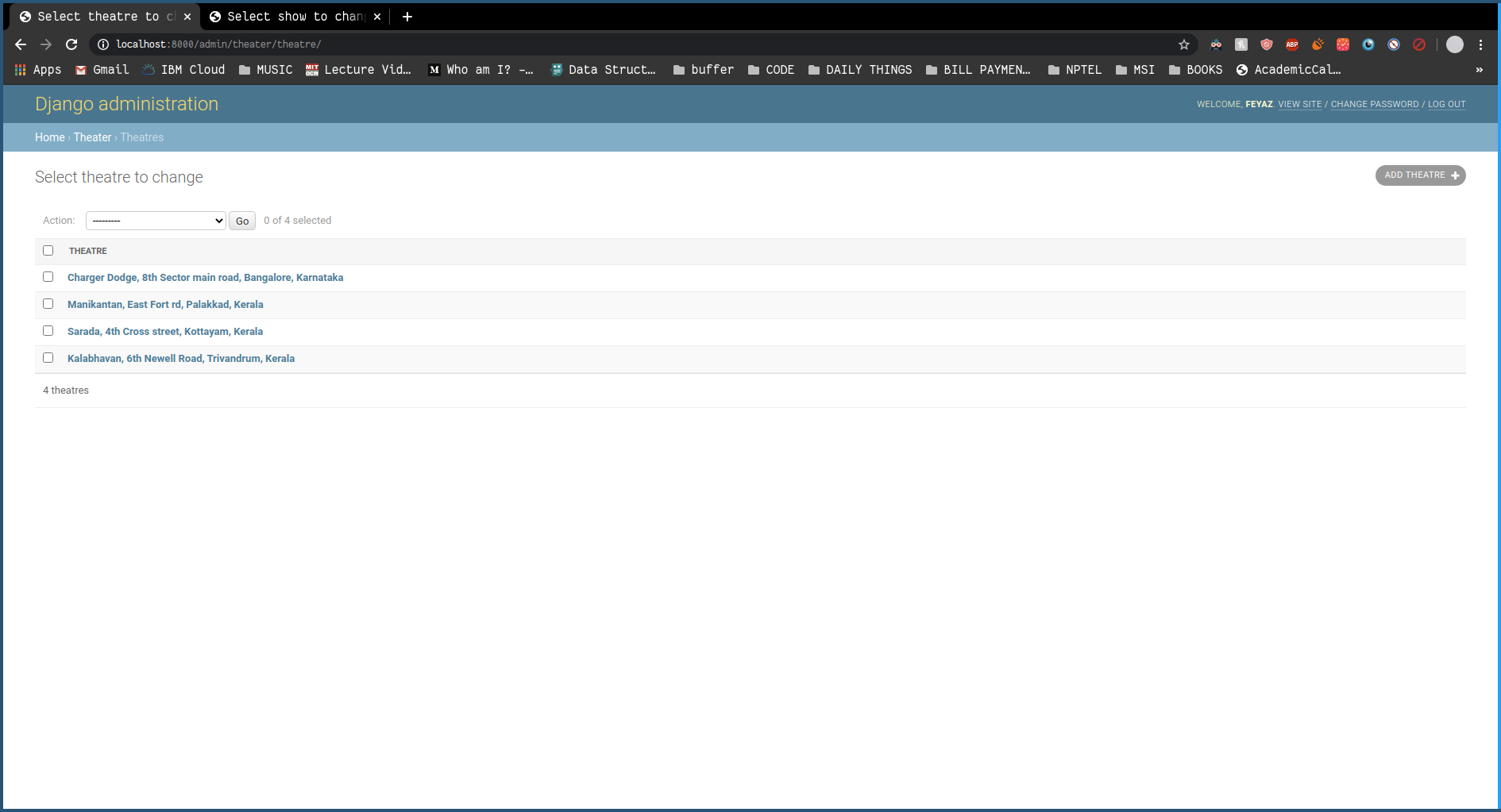
 Main Page

 Login Page

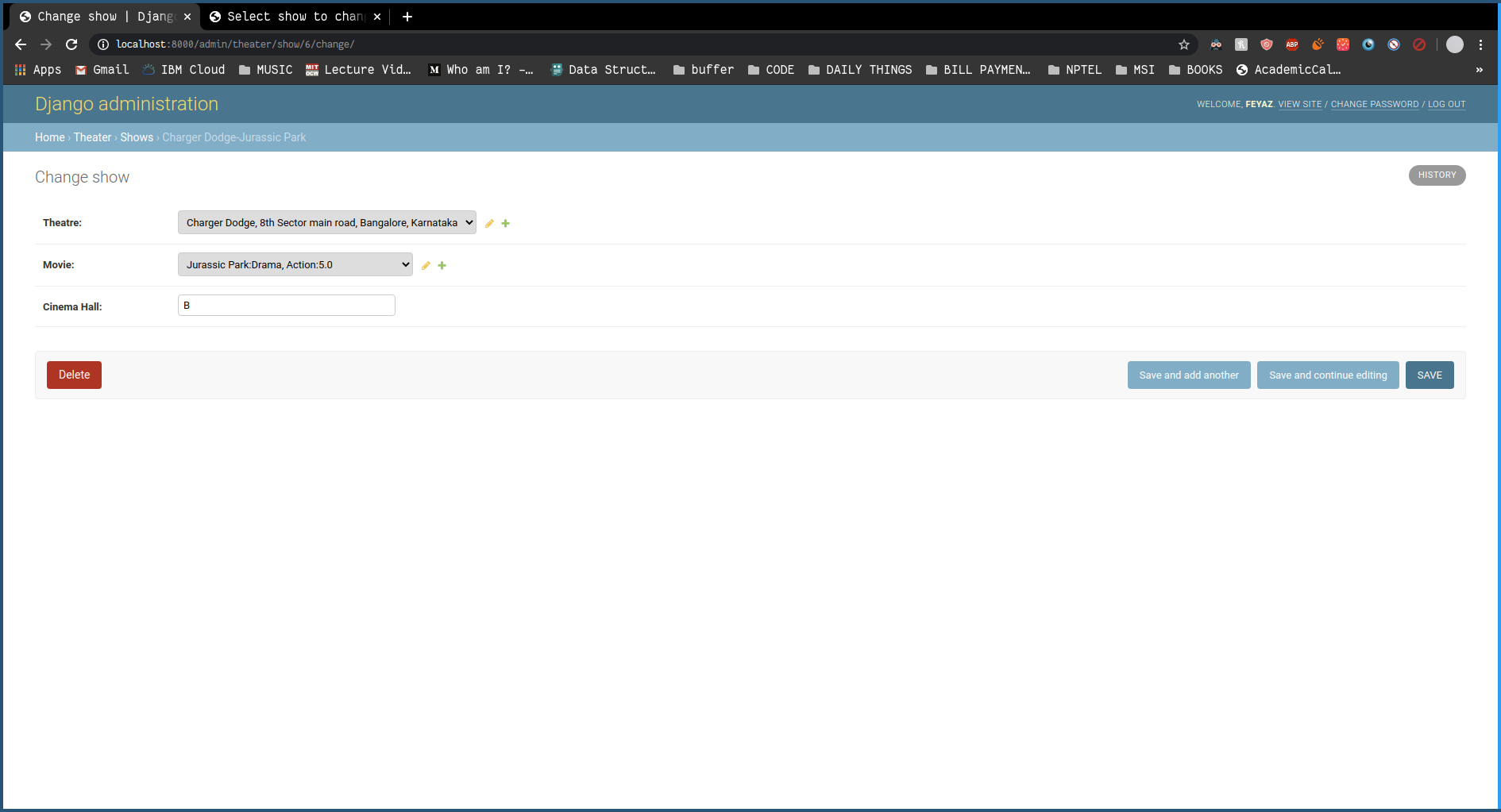
 Signup Page

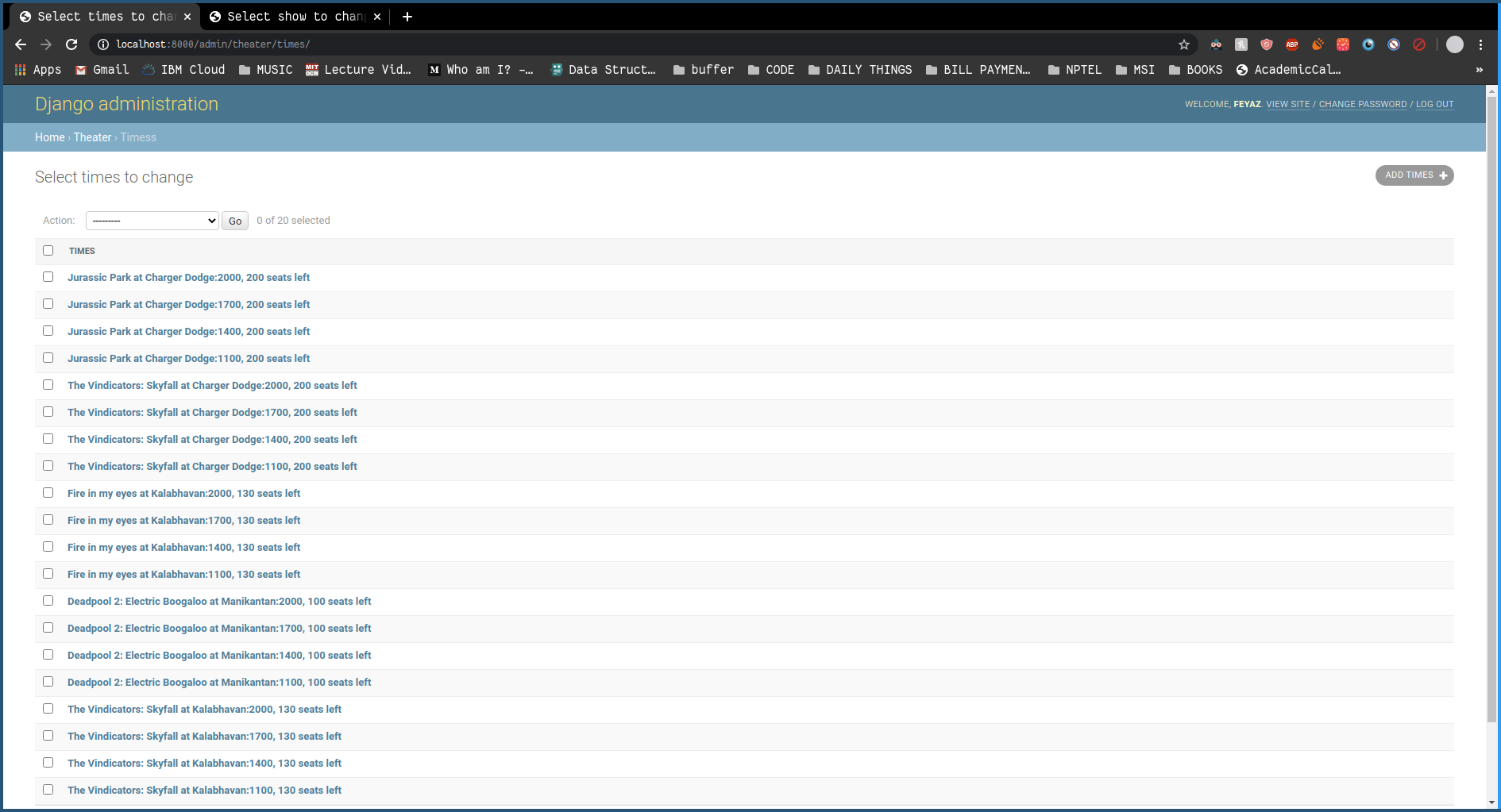
 Admin login page

 Add new movie

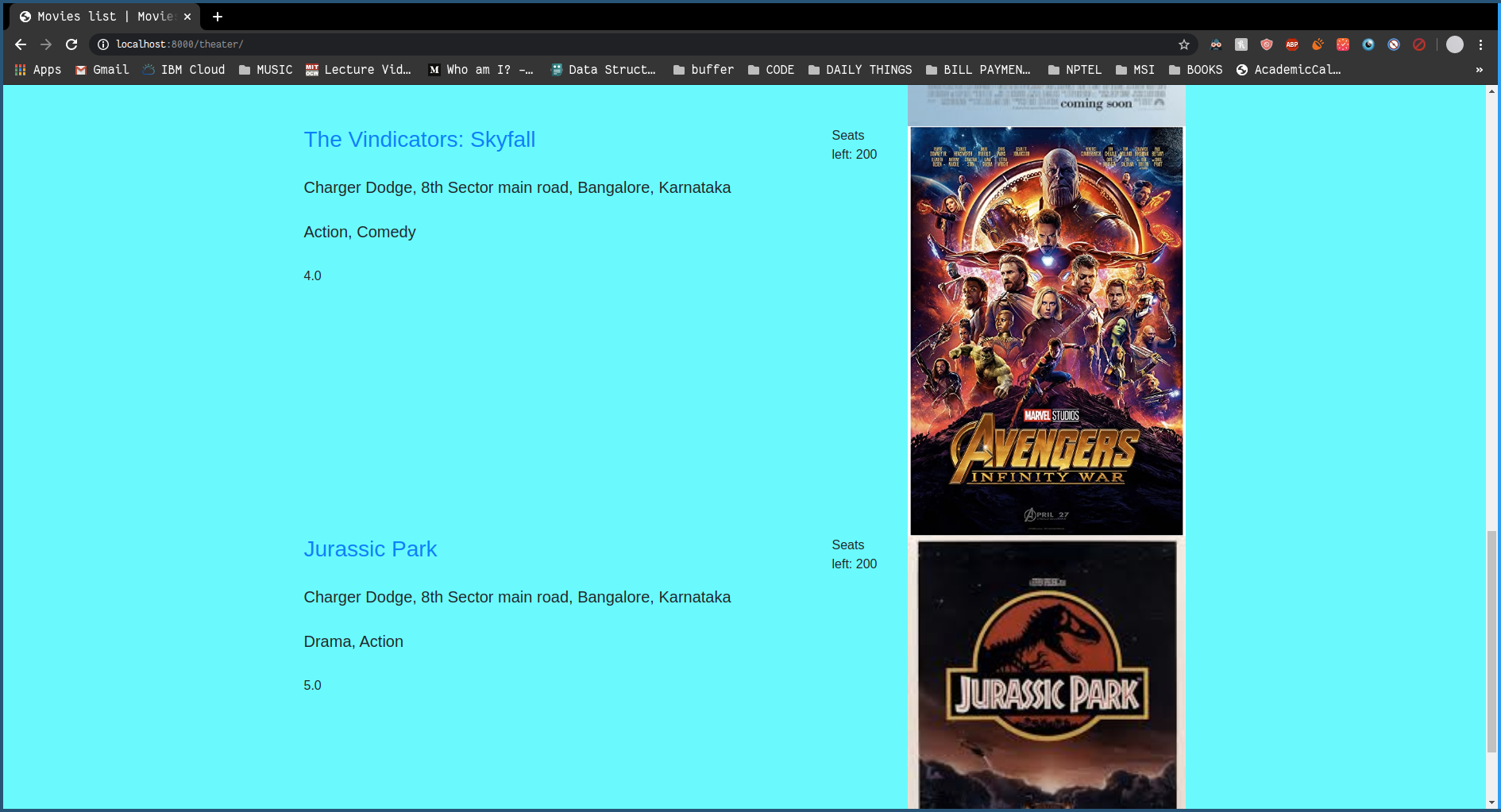


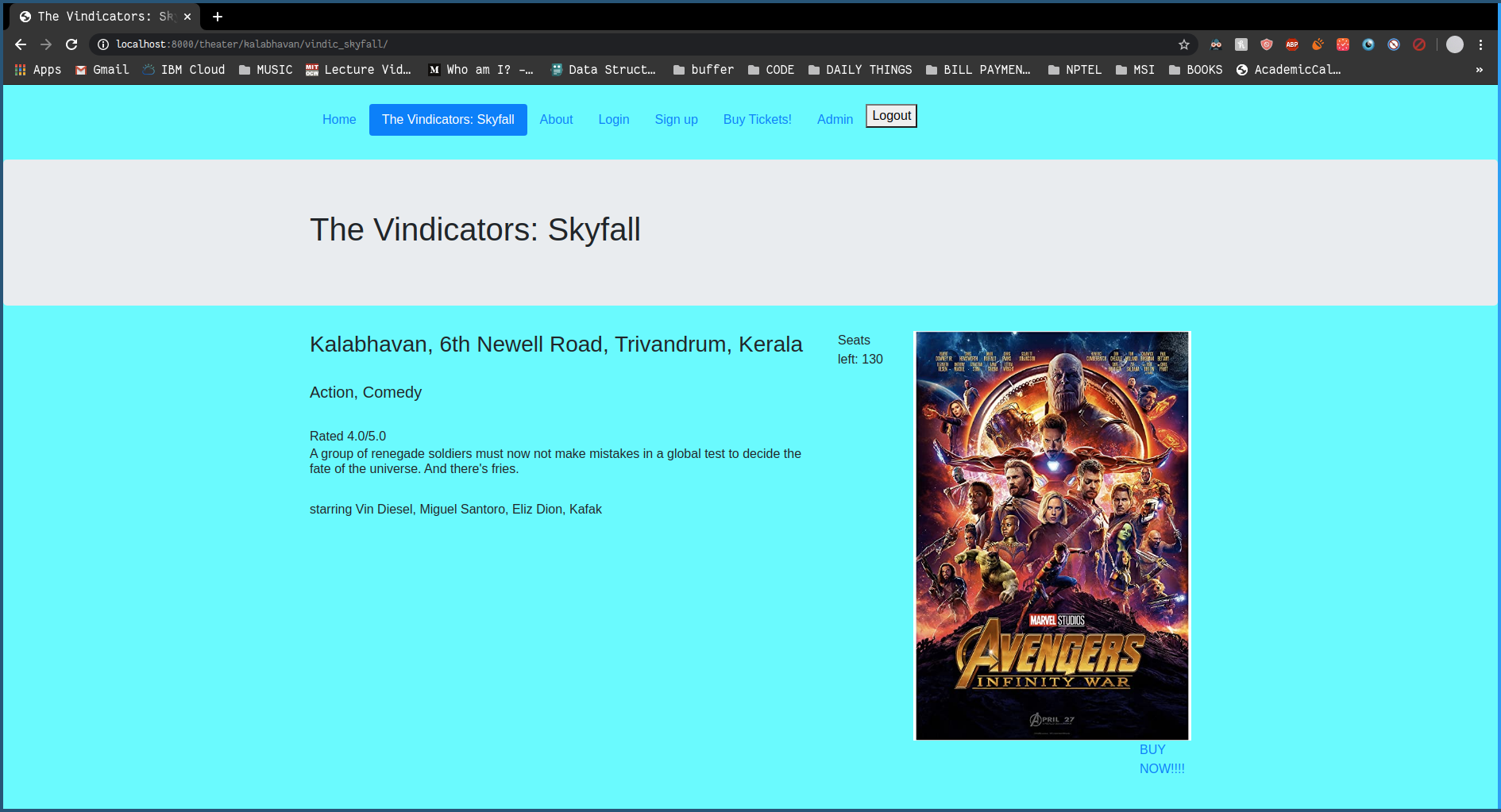
Theater list

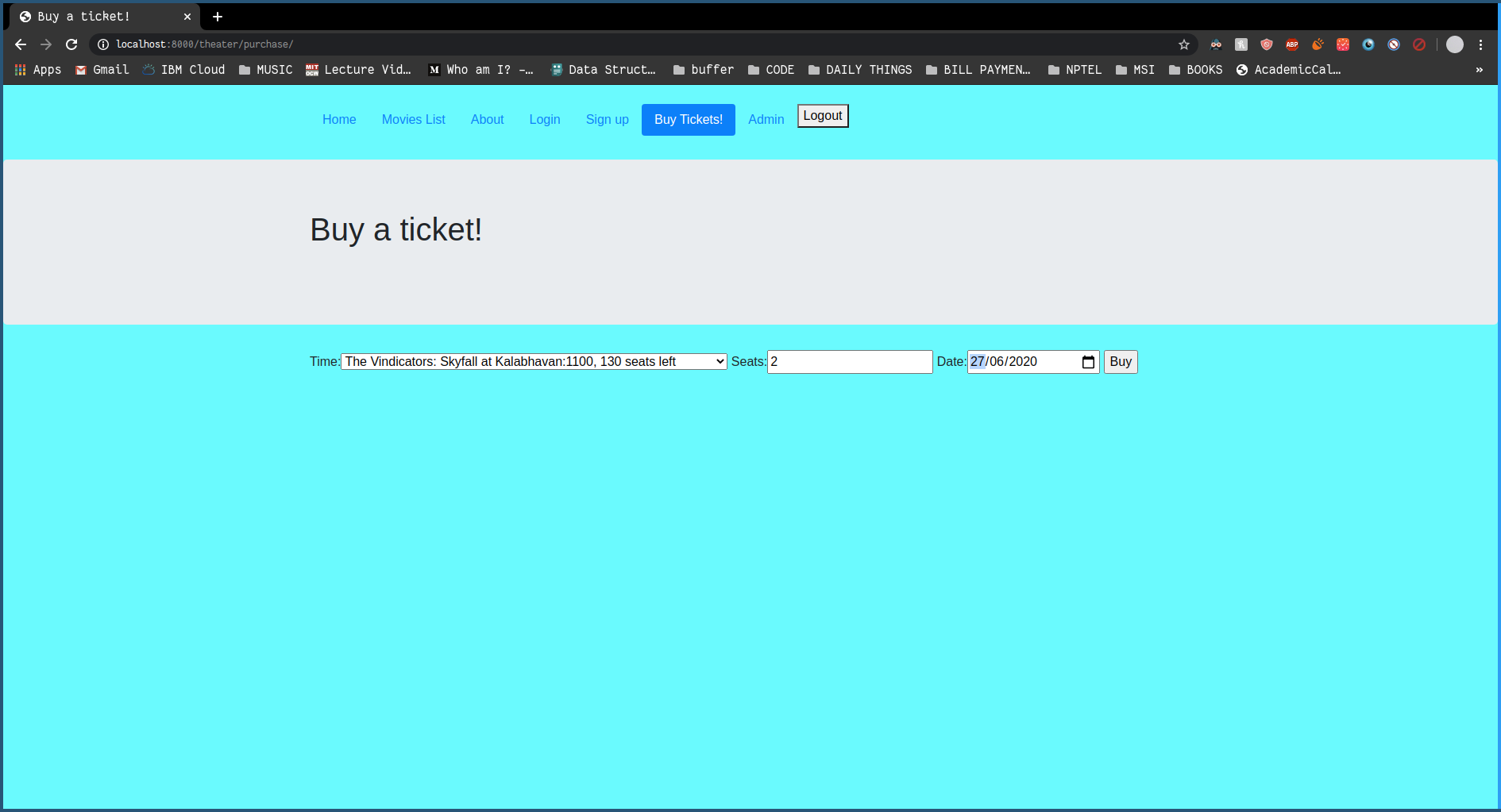
 Add show

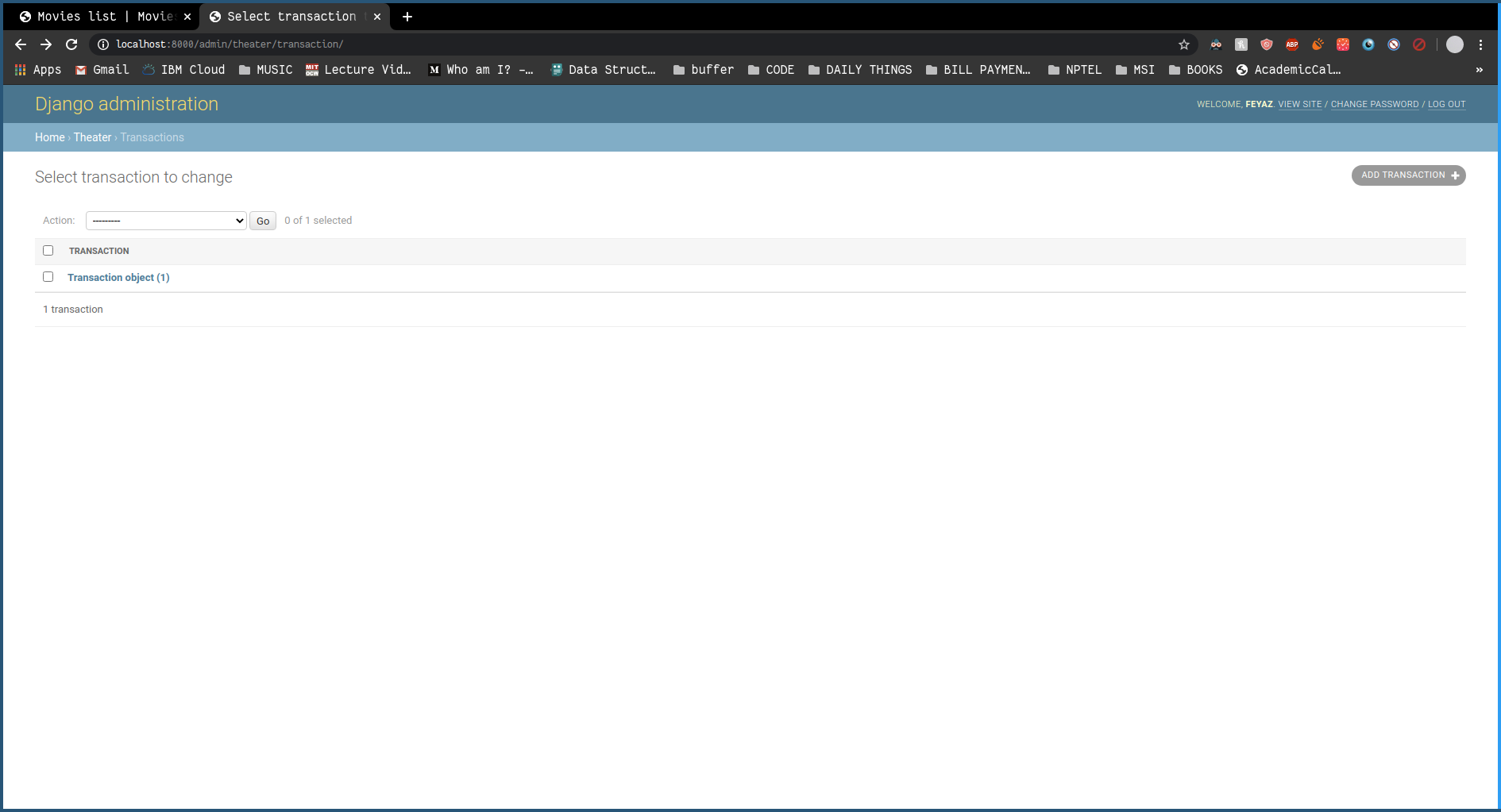
 Times table autopopulated

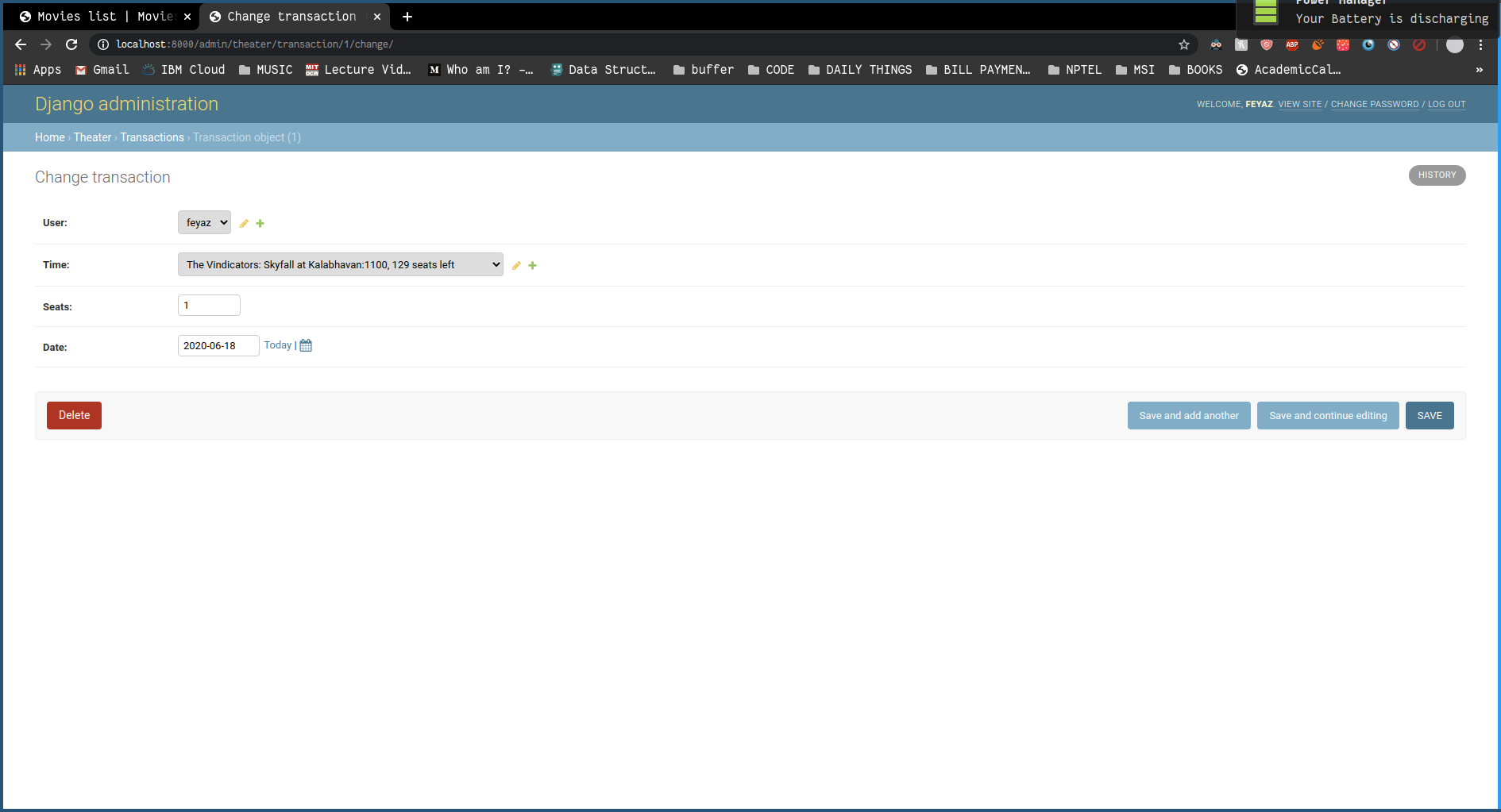
# 

 Now showing

 Show page for a movie at a theater.

 Buy ticket page

 New transaction table entry

 Transaction completed for user feyaz, at the specified show

# Implementation

## Why Django

The key to any project is the reliability and ease of use of the software used to implement it. For the overall design of our service we have utilized the Django framework, mainly due to prior experience in the software, to provide this data management solution to the existing flaws in the current systems.

We selected the Django framework since it is well documented and easy to learn so anyone willing to manage or work on our database would not find a massive barrier to entry. This was a key focus as we need the companies using this to require little to no extra training of staff on their part.

## How it was done

In standard Django fashion, we needed a name-space for separating different instances of functions, to prevent any ambiguity and ensure good coding standard. We started off by defining a name-space for the record-keeping features we would add, called ‘theater’. Django uses an object based storage model, making it easier to perform all CRUD operations with the ORM, Object Relational Mapping. This essentially means that instead of working through every reference of every object in the code, we could use a streamlined representation of all entries in every table, anywhere we wanted. First, we created each object, which would be stored in its associated table. We could later refer to the table by calling the model of that object we made. Django also automatically made an ID to act as primary key for its own query optimization, so all ER diagrams will reflect this. To properly separate all the pages we could show, we use views.

We knew concurrency would be a big issue in ticket booking, especially for movie fans and last minute rush. Django handles this part beautifully, with its own OSS transaction concurrency engine, which we have taken and tuned to reflect the needs of our specific database, like booking when all seats are taken, booking tickets for a movie that isn’t showing, or any such clerical error.

## Normalization

We had initially decided to implement according to the above ER diagram, but after learning about normalization we started to implement a few concepts from it. First we decided to add transactions to make it easier to track the tickets for each individual user. Then we made some more minor changes for ease of understanding.

As it stands, our Models are:

* User: A front end USER model for booking tickets.
* Admin: A USER model for making new theaters, editing show times and other admin level actions.
* Movie: An object for representing a specific movie.
* Theater: An object for a specific place that can show multiple movies, at different cinema halls.
* Show: An object for a specific Theater and a specific Movie, that shows which cinema hall is showing that movie.
* Times: An object for the different times at which a Show may be running, each with it’s own max capacity from Show, and an independent number of seats sold.
* Transaction: An object for tracking all transactions made on a specific Time, by a specific User, which automatically updates the seats sold for that Time.

Relations:

User: id, username, password\_hash

Movie: id, title, slug\_title, genre, air\_till, rating, starring, poster, blurb, Show

Theatre: id, theatre\_name, slug\_name, address, capacity, city, state, Show

Show: id, Theatre, Movie, cine\_no

Times: id, time, Show, capacity

Transaction: id, User, Time, seats

Functional Dependencies:

The candidate keys necessarily include id, as a function of Django’s ORM. No data has been repeated, and all relations use the object id as foreign key. Hence every candidate key is simply the id of each object. As such, there is only one candidate key, for each relation, from the id to every other attribute.

Normalization Checklist

1NF: No multivariate attributes

2NF: No partial dependencies

3NF: No transitive dependencies

BCNF: For every non trivial FD, X→Y, X is super key.

4NF: No multivalued dependency. (Our

5NF: Cannot be split into more relations without loss of data.

Hence, our database is in BCNF.

## A Deep-dive into the technical side

Django uses a Model View Controller (MVC) structure which allows easy access for admin users to alter required parts of the db. The MVC is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application.

These are the components of the MVC:

### Model

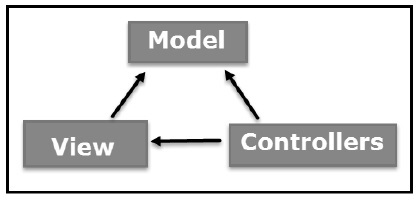
The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

### View

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

### Controller

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.



These were the core concepts used in the implementation of our project. This Controller allows easy access for users to changes the different relational models in the database.

After the database implementation we looked into the html part of it this was to make the interface for the program more user friendly and act as the starting point for our newly created databases.

## Requirements

The only requirements for the consumer would be a browser such as Google chrome. Same goes for the theater companies publishing their movies as the server side would be handled by the Database manager for this software.

# Conclusion

Our project is just how we feel we could use the beauty of database management to enhance the movie watching experience and make it as consumer friendly as possible cause we believe that at the end of the day the success of a product is determined by the satisfaction of the customer and we hope our project has begun to pave the way towards a more hassle free pocket friendly approach to the theater experience.

Our goals are set very high and this project is just a mere glimpse into the overall potential we believe this idea contains and hope to pursue it further with time.